Süsteemprogrammeerimine keeles C

Kuues loeng

Signaalid ja katkestused

- Probleem: arvutis ei juhtu asjad päris kohe
  - protsessor ei peaks riistvara järel ootama
- Kuidas lahendada?

Pärimine

- *Are we there yet?*
- Telefonitoru analoogia

Ligipääs katkestustele

- Kernel varjab riistvaralise kihi programmeerija eest
- Ometi on katkestused kasulikud ka meie jaoks

Blokeerimine

- Sisendi ja väljundi puhul jääb programm seisma: blokeeritakse
- Lihtne vältimine: loe ainult siis kui andmed on saadaval (kbhit() DOS/Win all)
- Keerulisem: (standard)sisendi blokeerimise mahalülitamine
- fd = open(„dev/ttyS0“), O_RDWR|O_NOCCTTY|O_NONBLOCK);
- ioctl(), fcntl()

Blokeerumine on vajalik

- Kui soovime et ka teised programmid paralleelis jookseks
- Lõimed võivad joosta (implementatsioonist sõltuvalt)
- select() funktsioon erinevate *socket*ite tarvis

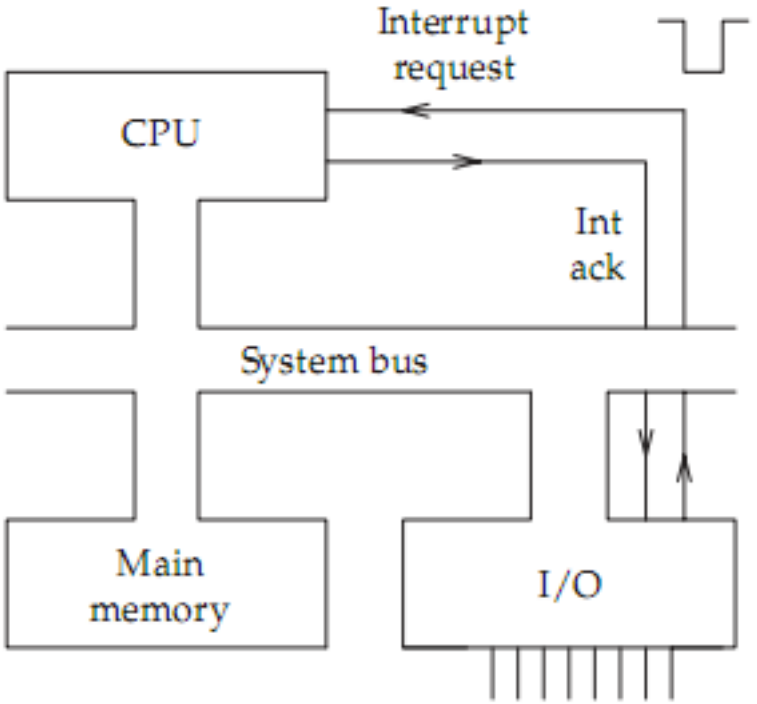
Katkestused

- Riistvaraline võimalus ühe sisendi muutumisel jooksutada mõnd konkreetset lõiku programmist
- Vajab riistvaralist tuge (üsna levinud)

Protsessoris

- Iga käsu puhul kontrollitakse ka katkestuste jalga
- Kui katkestus toimub salvestatakse käsupointer ja käivitatakse eraldi katkestuse teenindamise operatsioon (Interrupt Service Routine – ISR)
- Reageerimisaeg ~10µs

Joonis süsteemist



Exception vs Interrupt

- Katkestusi saab käivitada ka protsessorist seestpoolt
  - Protsessori veasituatsioonid
  - Mälukasutuse vead
- TRAP käsk tarkvarast

Allika tuvastamine

- Katkestusi on tihti ainult üks — kuidas tuvastada allikas?
  - Toosama küsitlemine?
  - Lisarauda ei vaja
  - Vähese hulga seadmetega mõistlik
- Vektorkatkestused
  - Vastuvõtul katkestanud seade annab endast ise märku või edastab teate järgmisele
- Programmeeritavad (PIC)
  - teab millisest tuli ja oskab tabelist öelda kuhu hüpe

Tegevustik

- Katkestus
- Protsessor salvestab programmiloenduri (PC) ja registrid stacki
- Interrupt Vector Table (IVR annab) ISRi algaadressi PC sisse kirjutamiseks
- ISR läheb käima

ISR

- Salvestab registri sisud
- Kontrollib allikat
- Eemaldab katkestuse põhjuse
- Taaskäivitab seadme?
- ...
- POP käsk stackile, RTE käsk PC endise seisundi taastamiseks

Kuidas programmeerijale samasuguseid võimalusi anda?

Signaal

- Signaali tüüpi kirjeldab täisarv.
- Kuna täisarvu meeldejätmine on väheotstarbekas, vastab igale signaalile mingi konstant (algavad nad prefiksiga SIG).
- Näiteks:
  - SIGKILL, SIGQUIT, SIGTRAP ja meie kõigi lemmik SIGSEGV - Segmentation fault

Signaalide väärtused (Solaris)

SIGHUP	1	Exit	Hangup (see termio(7I))
SIGINT	2	Exit	Interrupt (see termio(7I))
SIGQUIT	3	Core	Quit (see termio(7I))
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace or Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status Changed

+ veel

signal()

- Signaalidega ümberkäimiseks on järgmine käsk:

```
#include <signal.h>
void *signal(int signum, void (*handler)(int));
```

- signal() käsk määrab signaali numbriga signum saabumisel käivitatava funktsiooni, mis võib olla:
  - kasutaja poolt kirjutatud funktsioon
  - SIG_IGN : ignoreeri signaali
  - SIG_DEFL : taasta vaikimisi käitumine
- Täisarvuline argument *handler*ile on signaali number: ühe funktsiooniga saab hallata mitut erinevat signaali.

signal() (2)

- Signal taastab handleri eelneva väärtuse või SIG_ERR vea korral

```
extern int handler(int);
signal(SIGALRM, handler);
signal(SIGINT, SIG_IGN);
```

- Unix saadab protsessile signaali, kui midagi juhtub: näiteks kui aritmeetikas tekib ületäitumine, terminalil vajutatakse katkestusnupule või kui käivitatakse vigaseid käske. Protsessile saab signaale saata süsteemikäsuga kill()

kill()

```
int kill(pid_t pid, int sig);
```

- Võimaldab saata signaali igale protsessile või nende grupile
- Positiivse pid korral, saadetakse signaal vastavale protsessile
- Kui pid on 0, läheb signaal kõigile antud grupi protsessidele: näiteks ühelt terminalilt käivitunud protsessidele
- Kui pid on -1, saadetakse signaal kõigile protsessidele peale kõige esimese

kill() (2)

- Kui sa pole juurkasutaja, läheb -1 korral signaal ainult kill() käsku välja kutsuva protsessiga sama uid-ga protsessidele
- Kui pid on väiksem kui -1, läheb signaal kõigile protsessidele grupis -pid (**NB!** miinusmärk!)
- Kui sig on 0, ei saadeta signaali välja, aga veakontrollid samas töötavad
- Edu korral tagastub 0, vea korral -1 ja määratakse errno

alarm()

- Määrab kellaajalise signaali

```
unsigned int alarm(unsigned int seconds);
```

- seconds sekundi pärast antakse protsessile SIGALRM signaal
- Kui sekundid on 0, ei määrata uut alarmi
- Igal juhul tühistatakse plaanis olev alarm
- Tagastab mitme sekundi pärast eelmine alarm oleks toimunud, või 0, kui ühtki plaanis polnud

raise()

- int raise(int sig);
- Saadab vastava signaali protsessile
- Sealjuures Pythoni raise ning Java throw kasutavad sisemiselt sedasama meetodit veahaldusel

Handler

- Funktsioon, mis läheb katkestuse haldamiseks
  void nimi_siia(int sig)
- Saab argumendiks katkestuse numbri, ei tagasta midagi

Taktikad haldamiseks

- Haldamistaktikad
- Blokeerida teised katkestused haldamise ajaks
- Mitte kasutada funktsioone mis võiks olla samaaegselt käimas (re-entrancy, räägime lõimede juures)
- (1) Muuta mõnd globaalmuutujat ja seda programmis regulaarselt kontrollida
- (2) Vea puhul väljumine
    - haldus, handler tühistada, raise()

Signaalide blokeerimine

- Mõnikord ei taha me kriitilises olukorras signaale saada
- Blokeeritakse käsuga sigprocmask() (ja abistavad funktsioonid sellele nagu sigemptyset() jne

Lisaks

- sigaction()
- Veidi võimsam ja soovituslikum signaale haldav funktsioon
- Manuaal abistab

Signaalide ise saatmine

- Juhul kui ununes:
  - CTRL+Z : stop (pärast fg või bg käsk konsooli)
  - CTRL+C : interrupt (katkestus)
  - CTRL+\ : quit